Data Structures (CS 1520) Lab 8 Name:___________________________

Objectives: You will gain experience:

- get a feel for simple sorts: selection, bubble, and insertion sorts
- get a feel for advanced sorts: heap, quick, and merge sorts

To start the lab: Download and unzip the file: http://www.cs.uni.edu/~fienup/cs1520s19/labs/lab8.zip
The lab8.zip file you downloaded and extracted contains the following sorting algorithms which all sort in ascending order (i.e., from smallest to largest):

- bubbleSort.py - bubble sort code which **does not** check if it can stop early
- bubbleSortB.py - bubble sort code which stops early if no swapping is needed during a scan of the unsorted part
- insertionSort.py - the insertion sort
- selectionSort.py - the selection sort code we developed in class

Each program runs the sorting algorithm several time with different initial orderings of 10,000 list items. The initial orderings of items are: descending order, ascending order, random order, and random order again to check for consistence. Complete the following timings by running the each program.

Timings of Sorting Algorithms on 10,000 items (seconds)				
Type of sorting algorithm	Initial Ordering of Items			
	Descending	Ascending	Random order 1	Random order 2
bubbleSort.py				
bubbleSortB.py				
insertionSort.py				
selectionSort.py				

Study the code and answer the following questions about the sorting algorithms:

a) Why does the bubbleSort algorithm take less time on the ascending ordered list than the descending ordered list?

b) Why does the bubbleSortB algorithm take A LOT less time on the ascending ordered list than the descending ordered list?

c) Why does the insertionSort algorithm take A LOT less time on the ascending ordered list than the descending ordered list?

d) Why does the insertionSort algorithm take less time on the descending ordered list than the bubbleSort algorithm on the descending ordered list?

e) Why does the selectionSort algorithm take less time on the descending ordered list than the insertionSort algorithm on the descending ordered list?

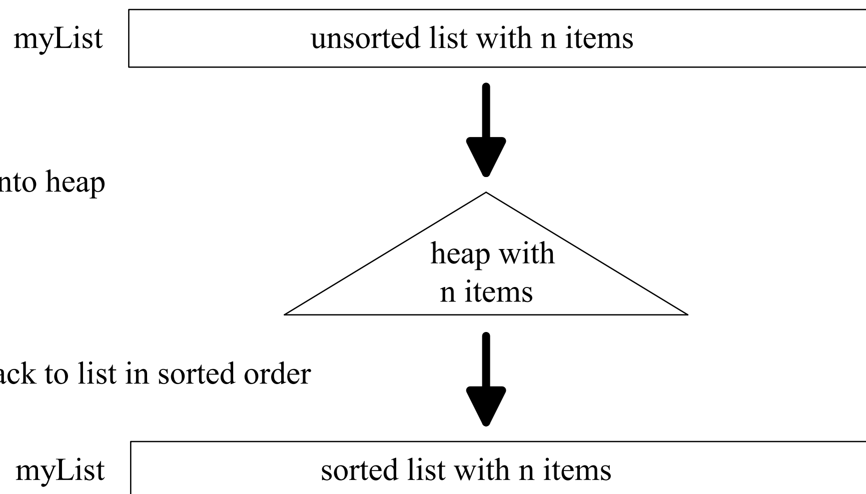
After you have answered the above questions, raise your hand and explain your answers.

Part B:

a) Complete the heap sort function in `lab8/heapSort.py` which contains the template for the heap sort algorithm discussed in class. Recall the steps of the algorithm:

Steps:

1. Create an empty heap
2. Insert all n list items into heap
3. delMin heap items back to list in sorted order

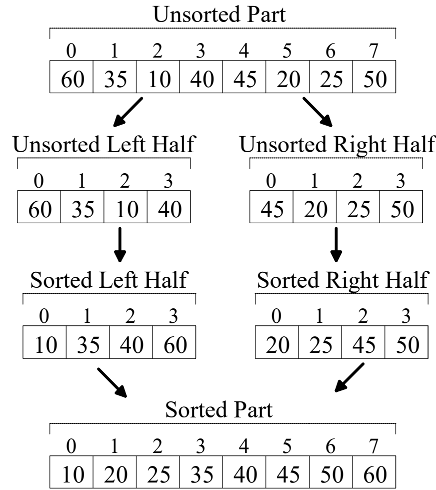


b) Time the heap sorting algorithm using `lab8/timeHeapSort.py` on 100,000 random items, 200,000 random items, and 400,000 random items.

# Items	Your Heap Sort Timing
100,000	
200,000	
400,000	

c) Explain the *O*( ) for your heap sort algorithm?

d) The general idea merge sort is as follows. Assume "n" items to sort.
- Split the unsorted part in half to get two smaller sorting problems of about n/2
- Solve both smaller problem recursively using merge sort
- "Merge" the solution to the smaller problems together to solve the original sorting problem of size n



The textbook's merge sort is in `mergesort.py`. Use the `timeMergeSort.py` program to run merge sort on a list of random items. Complete the following timing table:

Random # Items	Textbook's Merge Sort Timings
100,000	
200,000	
400,000	

e) Recall the general idea of *Quick sort* is as follows. Assume "n" items to sort.
- Select a "random" item in the unsorted part as the *pivot*
- Rearrange (called *partitioning*) the unsorted items such that:

- Quick sort the unsorted part to the left of the pivot
- Quick sort the unsorted part to the right of the pivot

The lecture 17 quick sort is in `quicksort.py`. Use the `timeQuickSort.py` program to run quick sort on a list of random items. Complete the following timings to get a feel for the "speed" of quicksort.

# Items	Lecture 17 Quick Sort Timings
100,000	
200,000	
400,000	

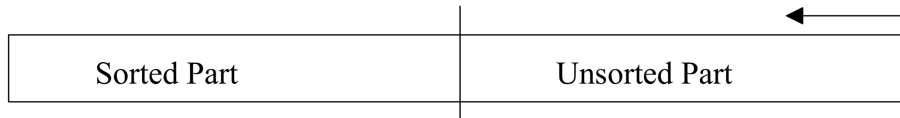
All three advanced sorting algorithms are $O(n\log_2 n)$ on initially random data. Why do you suppose quick sort is the fastest advanced sort on random items?

After you have completed the above times and answered the above question, raise your hand and explain your answers.

Part C: EXTRA CREDIT

a) Write (pencil-and-paper below) a variation of bubble sort that:

- sorts in descending order (largest to smallest)
- builds the sorted part on the left-hand side of the list, i.e.,



Inner loop scans from right to left across the unsorted part swapping adjacent items that are "out of order"

(Your code does NOT need to stop early if a scan of the unsorted part has no swaps)

```
def bubbleSortC(myList):
```

b) Implement and test your `bubbleSortC` code.